Docket No.: McIntoshProgrammerator

APPLICATION

FOR

UNITED STATES LETTERS PATENT

Title: Programmerator

Inventors: John W. McIntosh

Douglas P. Simons

Jonathan D. Gillaspie

Ray L. Bieber

BACKGROUND OF THE INVENTION

1. FIELD OF THE INVENTION

This invention pertains generally to computer systems, and more particularly to methods and techniques for monitoring, testing and controlling the operation of one or more computers through communications links.

2. DESCRIPTION OF THE RELATED ART

With the early development of computers, all processing capability was located at a single computer system. These early machines were massive structures using many vacuum tubes, the result of which was the generation of an enormous amount of heat, and an associated sensitivity to environment. With these massive structures, all processing was performed centrally at this main computer, owing to the substantial expense required for isolation and environmental control. While remote communications with a computer system were sometimes used, the use was extremely infrequent and necessarily limited owing to the poor communications capability available at the time. These limitations of the need for environmental control and lack of adequate communications capability each persisted for several decades.

Progress in the semiconductor industry, initially with compact calculators beginning shortly after 1970 and followed by much more concentrated and capable chips suitable for computers less than a decade later, diminished and has ultimately virtually eliminated the need for extensive environmental control. Likewise, communications equipment, protocols and bandwidth compression have opened up the ability for substantial remote communications that were inconceivable only a few

years ago.

For years, essentially from the days of first deployment of desktop computing, when a problem was encountered with a system, a computer user would be forced to resort to verbal telephone support with the hardware manufacturer. The waiting queues for these technical support personnel were notoriously long, with on-hold waits longer than an hour commonplace. When the technical support personnel were contacted, then the user would have to work verbally with the technical support person, and the support personnel would have to rely upon the computer users accurately describing the status and events, and performing operations requested by the support personnel. This arrangement was clearly less than optimum, requiring many times the effort that would have been required for the support personnel or a technician to directly diagnose and resolve the problem. Nevertheless, heretofore there has been little available for rapidly diagnosing the source of problems.

Unfortunately, many of the same issues and challenges face software vendors as those outlined above with regard to hardware manufacturers. When a particular program is prepared, the preparation work is usually performed upon a single type of computer having a particular combination of software installed thereon. All too frequently, the code will unintentionally rely upon components or features, such as may be found in the operating system, BIOS, system components or the like, which may vary from computer to computer. These variations may be based upon the release date of the particular computer, the software available at that time, upgrades provided by other vendors at the time of installation of their software, and other factors. At the time of deployment of early versions of the software, commonly referred to as alpha or beta versions, many of the incompatibility issues with diverse computers are discovered. Unfortunately, heretofore there

has been no efficient way to diagnose the incompatibility, nor to quickly test the computer or isolate the source of the problem. Help databases have been prepared where persons may look for similar problems. Nevertheless, the amount of time involved in isolating and diagnosing a problem is still enormous and a source of much waste in the industry.

Even during the development of the software, substantial testing must be done. As is known in the art of programming, while a change in one part of the source code may not be expected to have an effect elsewhere, all too frequently this expectation is incorrect. As a result, even the most minor changes require substantial testing and validation to ensure that the changes do not disrupt the performance of a program at any other point. Presently, many software companies employ persons specifically in the role of testing. These persons will be assigned the chore of interacting with the computer as though they were a regular user, trying out each of the functions and determining whether any bugs may be identified. This approach also requires substantial operation by testing personnel, and is somewhat unreliable owing to the difficulty in determining whether the testers are, in fact, completing the testing properly and thoroughly. Nevertheless, this approach still provides cost saving over discovering a problem in the field after the software or hardware has been released more generally. Furthermore, the reputation of the company is improved by having fewer problems with the released software or hardware than competitors who utilize less thorough testing.

In the area of system administration, similar problems are also encountered. An IT professional will typically be called upon to implement a new program, upgrade or other such tasks throughout an entire network or system. In such instance, the administrator will frequently be required to visit each and every computer in order to perform the necessary tasks, and to verify the proper functioning thereof. This opportunity to access the computers has been made far more

difficult with the advent of mobile systems and wireless communications, where many more of the computers connected through the network are not physically accessible at any given time.

In order to verify the performance of either software, hardware or a combination of the two, and regardless of whether the verification is being driven from the perspective of a manufacturer, developer, vendor, technical support, or internal maintenance within a single organization, this verification requires substantial interaction with the computer.

In an attempt to reduce the overhead associated with software debugging, a number of persons have developed methods for testing software by using a computer program. Many of these methods send information directly to the software or hardware, thereby bypassing the normal input channels and operations. Representative of the computer testing methods are U.S. patents 5,371,883 to Gross et al; 6,046,740 to LaRoche et al; 6,026,236 to Fortin et al; 5,022,028 to Edmonds et al; 5,249,270 to Stewart et al; 5,321,838 and 5,333,302 to Hensley et al; 5,335,342 to Pope et al; 5,594,892 to Bonne et al; 5,881,230 to Christensen et al; 5,926,638 to Inoue; 5,669,000 to Jessen et al; 6,119,247 to House et al; 6,195,765 to Kislanko et al; 6,249,882 to Testardi; 6,282,701 to Wygodny et al; and 6,353,897 to Nock et al; and 2002/0,099,978 to Kraffert, the contents of each which are incorporated herein for their teachings of the various methods and techniques associated with the control and operations associated with such systems. Nevertheless, no high level methods are introduced which readily permit the operator to perform the desired tests and operations on remote computers, particularly while interacting through the same or similar input devices and channels as would occur with standard human operations.

SUMMARY OF THE INVENTION

In a first manifestation, the invention is, in combination, a communications interface between a local computer and a remote system having a graphical user interface; a scripting language; and graphical user interface language extensions that enable the scripting language to control the remote system responsive to images appearing on the remote system graphical user interface.

In a second manifestation, the invention is a method for remotely testing the operation of a computer system. According to the method, a first element of the computer system graphical user interface is received. A user input action is then generated within the computer system responsive to the first element. The computer system graphical user interface is monitored for an expected second element within a predetermined time interval. A failure is signaled if the predetermined time interval elapses without detecting the expected second element.

In a third manifestation, the invention is a programmerator enabling a local system to remotely operate a computer system through local scripts and selectively respond to changes in graphical displays upon a graphical user interface of the remote computer system. A command capture interface displays a depiction of the remote system graphical user interface display and captures user input made therein. A command language set implements user input emulations representative of captured user input at the remote computer system and also implements image processing of the remote computer system graphical displays when processed by the local system. A scripting language has scripting commands that control a flow of execution of the local system in combination with command language set. An interface communicates between local system and remote computer system graphical user interface, responsive to the command and scripting languages.

OBJECTS OF THE INVENTION

Exemplary embodiments of the present invention solve inadequacies of the prior art by providing a high level interface between a computer and operator which will permit the computer to be controlled via a sequence of interactions. The computer may be monitored for display information which is expected, and also controlled in the event the expected information either is or is not obtained. The scripts may be executed through a remote interface, permitting remote testing, operation and evaluation.

A first object of the invention is to enable the execution of a plurality of user inputs directly through the computer system while responding to a video display. A second object of the invention is to enable the control of actions within the computer based upon the display. Another object of the present invention is to provide this capability using high level controls and commands that are intuitive and readily used. A further object of the invention is to enable this interaction and control to be instituted remotely through a communications interface through common Internet connection. Yet another object of the present invention is to enable both hardware and software developers to thoroughly test and evaluate the performance and operation of their developments and enhancements rapidly and with only minimal expense. An even further object of the invention is to permit a network or systems administrator to reliably, simply and quickly execute operations within an enterprise systems network, preferably without having to physically visit each of the computing devices within the network, and which would have been heretofore conducted with more time consuming, difficult and less reliable techniques.

BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing and other objects, advantages, and novel features of the present invention can be understood and appreciated by reference to the following detailed description of the invention, taken in conjunction with the accompanying drawings, in which:

FIG. 1 illustrates a preferred graphical user interface designed in accord with the teachings of the present invention.

FIG. 2 illustrates a preferred functional block diagram for the execution of functions designed in accord with the teachings of the present invention.

DESCRIPTION OF THE PREFERRED EMBODIMENT

Manifested in the preferred embodiment, the present invention provides a computer user the ability to control any other computer connected through an interface, using the standard graphical user interface (GUI). As illustrated in Figure 1, a user interface which takes the form of an interactive development environment (IDE) 100, will preferably include several functional windows 110, 120, 130 therein. More particularly, window 110 provides a command capture interface which most preferably provides a system-under-test (SUT) screen display 114 which provides a visual depiction of the display screen generated by the system for which the operation of the interactive development environment 100 is designed to control. Within that display there will be various graphical representations 116 that may take the form of icons as illustrated, or various buttons, message boxes, prompts or any other graphical image or control. The depiction may be an exact copy of the remote graphical user interface, a resized or scaled version, or may further include various image manipulations such as color conversions or other manipulations as deemed suitable

and appropriate for a given application. A command tool bar 112 is provided which allows a user of interactive development environment 100 to select what type of command or action will most desirably be implemented at any given state within the system-under-test screen display 114. These commands will most desirably replicate the functions at the system-under-test 290 as though they were, in fact, executed directly upon that system. Preferably, such commands will include keyboard commands and mouse commands, though it will be understood that any form of user input may be emulated. Consequently, touch screen monitors, graphics pads or tablets, and any other type of primary or peripheral input device may be emulated as required and designed for. In the case of the keyboard commands, two commands may, for exemplary purposes only and not limited thereto, be implemented. The commands include a "TypeText" command and a "TypeCommand" command. These two commands permit any keyboard input available to be implemented, and yet provide a very understandable means to implement the same. In other words, where simple text must be entered, the "TypeText" command will be utilized. Where function and command keys are to be implemented, the "TypeCommand" function may be used. These commands are most preferably higher level language commands which will later be processed by a specific language extensions processor 255 described in greater detail herein below with regard to figure 2. Similarly, such mouse functions as "Click", "DoubleClick", "RightClick", "MouseButtonDown", "MoveTo", "Drag", and "MouseLocation()" may be implemented. For the purposes of this disclosure, and as is commonplace in the computer arts, it will be understood that the use of parentheses denotes the implementation of a function that may include data or other variables that are being passed either from or to the function. Special commands such as "ClickAny", the operation which will be described herein below, may also be implemented.

Operation of the present invention depends upon the graphical user interface. The myriad of possible functions and displays that may be produced as a result of an operation are extensive. For example, there are times where an operation may result in an introduction of one of a variety of controls. Where a single entity is anticipated, the image of the entity can be stored through the command capture interface window 110, and, when such image is later detected during execution of a script, an appropriate action can be selected, such as the "Click" command which would represent a mouse click upon a particular point in the screen. While many times this "Click" command may be executed directly upon the entity which newly appears upon the system-under-test screen 114, the direct action upon such an entity is not required. Instead the user of interactive development environment 100 has complete control over any of the user actions that may be relayed to the system-under-test 290, such as providing typed text, commands, movement of the mouse, and so forth. Consequently, the appearance of an object may stimulate any suitable action. Where such action would be to send a click on any active component that may appear on the screen, the "ClickAny" command will most preferably be provided, which enables the click to occur on any entity that may appear. Furthermore, in one contemplated embodiment, the activation of keystrokes or mouse clicks directly within the system-under-test screen 114 may be directly transferred as commands that are captured, or there may alternatively be a button to select which initiates and subsequently stops the recording of such within system-under-test screen 114 actions.

Since the present invention is designed to control graphical user interfaces, several commands are contemplated herein, but once again not considered to be solely limiting or restricted thereto. These image commands will most preferably include screen image searching commands and specific image information. Exemplary of the screen image searching commands are such

commands as "WaitFor", "WaitForAny", "WaitForAll", "RefreshScreen", "ImageFound()", "AnyImageFound()", "ImageLocation()", "AnyImageLocation()", "EveryImageLocation()", and other similar commands. A variety of information will most preferably be obtained or obtainable with regard to specific images, through such commands as "ImageInfo()", "FoundImageNumber()", "FoundImageName()", "FoundImageLocation()" "ImageHotSpot()", and "ImageSize()". Utilizing the above command set, it is possible to monitor a graphical user interface for any type or shape of image and then, responsive to the presence thereof, select a subsequent user action as though the user action were being performed directly upon the system-under-test 290 rather than from a source or controlling computer. In the event an unexpected event or entity appears upon the screen, the user, through the integrated development environment 100, has the opportunity to control the operation of the local and remote systems responsive thereto. For example, if a particular object is expected to appear upon the system-under-test screen 114 within a particular time interval, and the time expires prior to the object appearing, then it would be possible for a script entered within script window 120 to time out and cause an error message or warning message or the like to appear upon the local computer screen. The scripting window will most preferably provide access to functions and commands through, for exemplary purposes only, script tool bar 122 that are commonly associated with or available through any of the various scripting languages or more fundamental programming languages. Such functionality as decision structures, loops, timers, and the various other myriad of functions available therein as are well known will most preferably be incorporated herein, in accord with the particular language incorporated herewith or developed for operation herein. One such example is "SenseTalk", though other suitable scripting languages are certainly contemplated herein, and will be dependent upon the preferred operating platform or cross-platform

capability desired.

Most preferably, interactive development environment 100 will additionally include a window 130 which provides access to various organizational functions, which may, in the preferred embodiment, be implemented using the familiar index tab analogy. Among the available index tabs may, for exemplary purposes only, be functions such as the storage of various scripts 132, images 134, results from past script executions 136, scheduling of planned script executions 138, and the identification and or the location of helper script and image files 140, which may be local, on a network, or located anywhere on the Web.

While figure 1 provides a basic overview of the user view provided through the interactive development environment 100, figure 2 illustrates the functional interconnection of the interactive development environment 100 components with the various additional components that are not visible upon the screen to form the preferred programmerator system 200. More particularly, the presentation of the system-under-test screen 114 is achieved by a remote GUI interface 270. A command and image capture system 212 is responsible for the capture of appropriate commands from command tool bar 112 and image information such as icon 116. Consequently, when a user selects a particular command, such as a mouse click or the like, this command must be captured and incorporated into the appropriate scripting information. Likewise, the inclusion of an expected image or others of the herein above described image or user control commands must be incorporated. These commands, which are not native to prior art operating systems, programming or scripting languages, are passed through to the system data, scripts and images controller 235. Controller 235 is responsible for the appropriate redirection and incorporation of command, graphics and scripts between execution environment 245, script window 120, and command and image capture system

212. For example, when a command is received from command and capture system 212, this command and any associated language will preferably be inserted into scripting window 120. Likewise, the passing of images from the execution environment 245 will occur through controller 235. Script window 120, which would in operation contain an active script, may be processed directly through execution system 240 for a real-time run, or may be passed through organizational window 130 for storage and scheduling therein.

Ultimately, a given script will be executed through the execution system 240, which is configured to carry out the processes specified within a script. While somewhat simplistic in description, the execution system 240 and execution environment 245 are typically comprised by the local CPU, memory, OS, and the like. The command processor or CPU will effect or control much of the execution within system 240, but will be monitoring a diverse set of status indicators, potentially both locally and at the system-under-test 290, programming calls, and the like. These various items being monitored comprise in part the execution environment 245.

As the script is being processed, execution environment 245 will need to call the scripting language processor 250, which may be an interpreter, compiler or other suitable language processor. The scripting language has been extended in the preferred embodiment by various GUI commands and controls that are created through the command capture interface 110, such as the various mouse events and the like. Consequently, these commands must be processed not by the scripting language processor 250, but instead by a language extensions processor 255. As a result of the processing of the scripting language and the language extensions, an action may require to be initiated, which would be detected and triggered in the initiate action subsystem 260, which will relay the action back to execution environment 245. In the event this is an action intended for the system-under-test 290,

such action will be relayed from execution environment 245 through remote GUI interface 270. The purpose of the remote GUI interface 270 is to implement an interface with the remote system-under-test 290, preferably relatively independently of the characteristics of communications channel 275 used and the data exchange rate associated therewith. This consists of providing client function to facilitate communication with the GUI server on the remote system-under-test 290, and to implement any needed out-of-band communication. Included are such operations as the client-server functions of retrieving images, sending keyboard and mouse commands, and initiating error recovery procedures as needed.

Communications channel 275 will in the preferred embodiment include a keyboard channel 272, mouse channel 274, and a transfer of screen updates from VNC server 280 back to the remote GUI interface 270. Communications channel 275 may be a high speed trunk line or cable, or may alternatively be a relatively slow-speed dial-up or RS-232 type connection. With proper selection of components, the preferred embodiment has much flexibility to operate through diverse communications channels having very different data transfer rates and signal to noise ratios.

To achieve broader application to more and more diverse systems-under-test 290, remote GUI interface 270 through a communications channel 275 communicates with remote computer virtual network computing server 280 or the like. As illustrated herein, the remote GUI interface 270 and VNC server 280 are most preferably VNC components which are readily available commercially and which operate as cross-platform components to directly interface with a remote system GUI. Nevertheless, other standard interfaces may be supported.

Images from the system-under-test 290 GUI will be relayed through VNC server 280 or the equivalent back to local system remote GUI interface 270, and from there routed to the GUI

recognition subsystem 265. GUI recognition subsystem 265 dynamically scans the screen image of the remote system-under-test 290 for any bit-map images which the initiate action subsystem 260 is searching for. The goal of GUI recognition subsystem 265 is to locate images and signal the presence of such images to the initiate action subsystem 260 through an image detection signal, initiate recovery procedures such as moving the mouse cursor to locate images, or to report that the sought-after images are not available. GUI recognition subsystem 265 cooperates with the initiate action subsystem 260 and language extensions processor 255 output to determine whether a desired event or image has been created and if so, to execute the desired image identification actions requested within the language extensions received from language extension processor 255 through the execution environment 245.

Initiate action subsystem 260 initiates any action requested by the language extensions received from language extensions processor 255, which could be active commands to type text through the keyboard or to move the mouse in a number of ways. The commands may also in the preferred embodiment include passive commands to search the screen image on a continuing basis for a specific bit-map image, and, for exemplary purposes, terminating the search after a given period of time has elapsed.

While the preferred embodiment programmerator system 200 depends upon GUI image location, this may be practiced in one of several different ways. One approach is to directly map bits and to locate based upon this matching bit pattern. Said another way a black border with a blue interior and a yellow button bar may comprise the image of interest, which might, in this case, bee a control or graphic. However, when some potential systems-under-test 290 are configured with a different color scheme, the same control may be comprised of a different color set and would, using

this simpler approach, go undetected. Several approaches may be used to overcome this limitation. One such approach is to develop a custom color scheme at the local computer, in effect translating the remote system-under-test 290 display to the local color scheme. This may, for example, be achieved by access the system-under-test 290 and determining the color scheme being implemented within the operating system or the like to determine what color translations are required to restore color variations to a common baseline. Another approach is to map and convert the images based upon geometry into particular color sets. Yet another approach is to convert to black and white colors. Depending upon the design and intent, and the types of systems to be tested or controlled, one or more approaches may be provided or available within the operation of the programmerator system 200.

While the foregoing details what is felt to be the preferred embodiment of the invention, no material limitations to the scope of the claimed invention are intended. Further, features and design alternatives that would be obvious to one of ordinary skill in the art are considered to be incorporated herein. The scope of the invention is set forth and particularly described in the claims hereinbelow.